COBB
INSIDE VISUAL dBASE™

May 1996 • Vol. 3 No. 5
US $8.50

Tips & techniques for Visual dBASE and dBASE for Windows

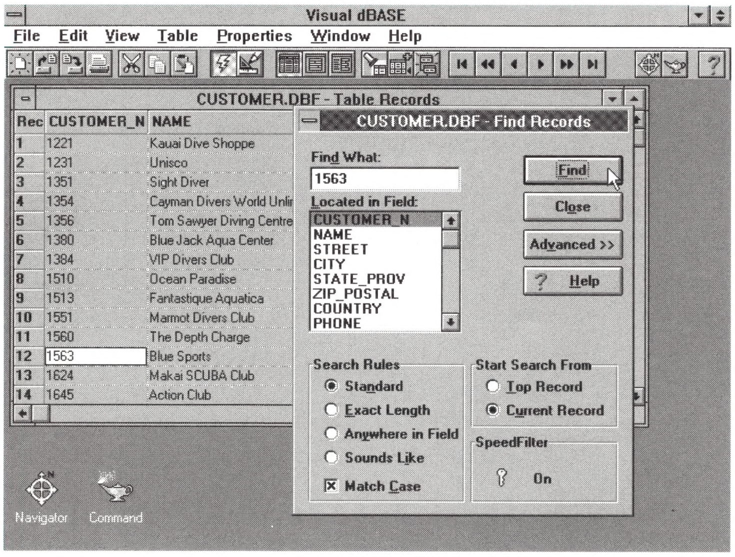
Tools

Adding a Find Records dialog box to your applications

by Keith Chuvala

Finding records in Visual dBASE is easy. You just open the Find Records dialog box in the Table Records window (also known as the BROWSE screen), as shown in **Figure A**. This handy tool pops up when you press [Ctrl]F, press the Find Records button on the toolbar, or select Find Records from the Table menu. It's a handy dialog box, because the user can execute almost any kind of LOCATE command with absolutely no knowledge of dBASE syntax.

Figure A


Visual dBASE
File Edit View Table Properties Window Help
CUSTOMER.DBF - Table Records
CUSTOMER.DBF - Find Records
Find
Close
Advanced >>
Help
Navigator
Command

Rec	CUSTOMER_N	NAME
1	1221	Kauai Dive Shoppe
2	1231	Unisco
3	1351	Sight Diver
4	1354	Cayman Divers World Unlir
5	1356	Tom Sawyer Diving Centre
6	1380	Blue Jack Aqua Center
7	1384	VIP Divers Club
8	1510	Ocean Paradise
9	1513	Fantastique Aquatica
10	1551	Marmot Divers Club
11	1560	The Depth Charge
12	1563	Blue Sports
13	1624	Makai SCUBA Club
14	1645	Action Club

Find What: 1563

Located in Field: CUSTOMER_N, NAME, STREET, CITY, STATE_PROV, ZIP_POSTAL, COUNTRY, PHONE

Search Rules: [✓ Standard] Exact Length Anywhere in Field Sounds Like [✓ Match Case]

Start Search From: Top Record [✓ Current Record]

SpeedFilter: On

The built-in Find Records dialog box is available only in the Table Records (BROWSE) window.

IN THIS ISSUE

We welcome a new editor-in-chief!

Welcome to *Inside Visual dBASE!* This month, we introduce a new editor-in-chief—Keith G. Chuvala. As a long-time member of Borland's TeamB, Keith has provided invaluable support for dBASE programmers. He has authored a number of books about dBASE programming and is a professor of computer science at Southwestern College in Winfield, Kansas. You can reach Keith via his Web page at

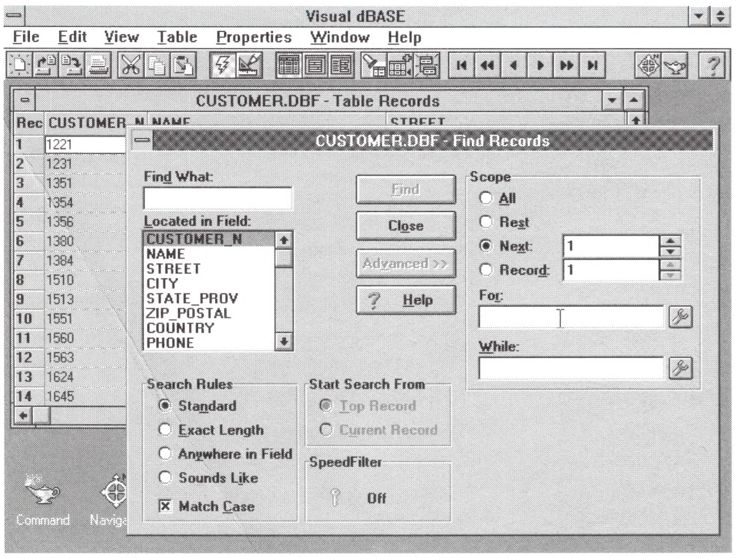
`http://www.sckans.edu/~kgc/`

You can also write to *Inside Visual dBASE* at The Cobb Group's E-mail and snail mail addresses, which are listed in the masthead on page 15. We look forward to hearing your suggestions and comments for *Inside Visual dBASE*.

As we welcome Keith, we bid a fond farewell to Peter Schickler. Peter has reopened his Xbase consulting business in Castleton, Vermont, and we wish him the best!

The Advanced pushbutton reveals controls for the scope of the search to be performed, as shown in Figure B. Of particular interest in this extended form are the For and While text boxes. These options allow the user to specify almost any criteria imaginable. Although these options might seem a bit esoteric for a simple Find Records function, you'll discover that power users really appreciate this flexibility. In this article, we'll show you how to add your own Find Records dialog box to the applications that you distribute to your end users.

Figure B


Visual dBASE
File Edit View Table Properties Window Help
CUSTOMER.DBF - Table Records
CUSTOMER.DBF - Find Records
Find
Close
Advanced >>
Help
Command
Naviga

Rec	CUSTOMER_N
1	1221
2	1231
3	1351
4	1354
5	1356
6	1380
7	1384
8	1510
9	1513
10	1551
11	1560
12	1563
13	1624
14	1645

Find What:

Located in Field: CUSTOMER_N, NAME, STREET, CITY, STATE_PROV, ZIP_POSTAL, COUNTRY, PHONE

Search Rules: [✓ Standard] Exact Length / Anywhere in Field / Sounds Like; Match Case [✓]

Start Search From: [✓ Top Record] Current Record

SpeedFilter: Off

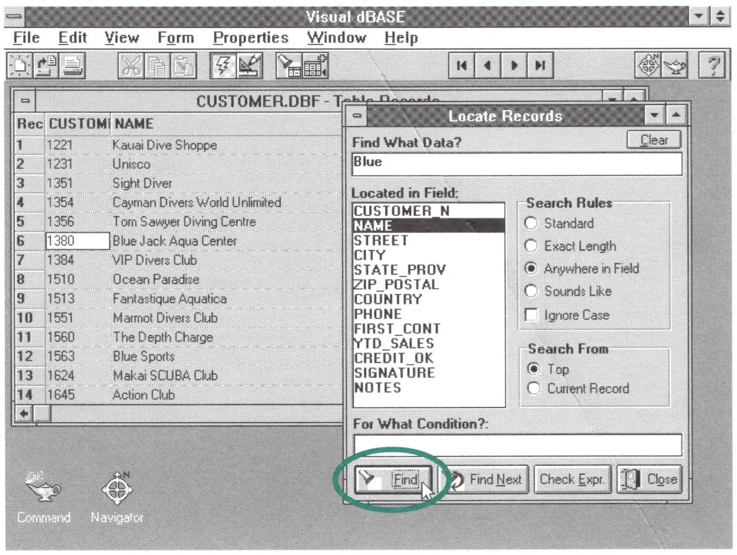
Scope: All / Rest / [✓ Next: 1] / Record: 1

For:

While:

When you click Advanced in the Find Records dialog box, you can specify scoping rules and global conditions for the search.

Figure C



Rec	CUSTOMI	NAME
1	1221	Kauai Dive Shoppe
2	1231	Unisco
3	1351	Sight Diver
4	1354	Cayman Divers World Unlimited
5	1356	Tom Sawyer Diving Centre
6	1380	Blue Jack Aqua Center
7	1384	VIP Divers Club
8	1510	Ocean Paradise
9	1513	Fantastique Aquatica
10	1551	Marmot Divers Club
11	1560	The Depth Charge
12	1563	Blue Sports
13	1624	Makai SCUBA Club
14	1645	Action Club

Find What Data? Blue

Located in Field: CUSTOMER_N, [✓ NAME], STREET, CITY, STATE_PROV, ZIP_POSTAL, COUNTRY, PHONE, FIRST_CONT, YTD_SALES, CREDIT_OK, SIGNATURE, NOTES

Search Rules: Standard / Exact Length / [✓ Anywhere in Field] / Sounds Like; Ignore Case [ ]

Search From: [✓ Top] Current Record

For What Condition?:

FINDBOX.WFM produces a dialog box that resembles Find Records but that's streamlined for ease of use.

Great tool, but no handle!

You and your users may agree that the Find Records dialog box is a great tool. Unfortunately, you can't call this dialog box directly from a dBASE application. However, the Find Records dialog box *is* available through the menus of an MDI application if (and only if) you use a BROWSE command in your code, *and* you don't use your own menu file.

Suppose that, in your applications, you don't use BROWSE at all. Instead, you construct forms containing Browse control objects. In that case, you'll discover that the Find Records dialog box doesn't socialize with those controls. Therefore, Find Records is something you're not normally going to be able to include in your running applications.

Reinventing the flashlight

You may find that sometimes you simply must write some code to get the job done. In order to incorporate a Find Records dialog box into your application, you can use FINDBOX.WFM, shown in Listing A on page 4. FINDBOX.WFM is a home-grown version of Find Records. Instead of replicating every last detail of the built-in Find Records dialog box, we've condensed it to include only the functions that users have found valuable in our custom applications. FindBox, shown in action in Figure C, looks and acts much like the built-in Find Records dialog box, so users familiar with the built-in tool will quickly feel at home with our new, streamlined version.

Most of the differences between FindBox and the built-in tool are evident in the custom dialog box's slimmer appearance. We chose to include only the For control (which we've named "For What Condition?") from the built-in tool's Advanced section. Even the more sophisticated users who run our applications report that they use For frequently, but they almost never use While.

Users report that the Find Next button is another welcome addition, despite the fact that you can achieve the same effect by using the Search From Section's Current Record radio button. The Find Next feature is just a more natural and intuitive way to continue the search for the next occurrence of the specified data.

The Find Next approach also plays well into the scheme of the dBASE code in our form. Specifically, we make the programming simple by using a basic CONTINUE statement to repeat the last LOCATE from the current position.

Performance is (usually) only skin deep

Internally, the FindBox routine is straightforward. You construct a LOCATE clause based on the values of the various controls. The O'clock event handler for the Find button does this job and also executes the LOCATE command itself. You trap errors in a rather generic way and report errors via a message box should a problem occur.

LOCATE doesn't usually work as fast as SEEK or FIND, which both use an index. Visual dBASE's SpeedFilter technology sometimes steps in to help the LOCATE command, though, and when it does, it delivers terrific performance.

The built-in Find Records dialog box displays a cute little key icon to tell you when your custom condition is appropriate for SpeedFilter to take over. However, many users don't understand the difference, and most of them don't even care.

Notes from a live application

I originally added the Check Expr (check expression) pushbutton for use in testing the custom Find Records program. Its OnClick event handler simply displays a message box with the LOCATE expression created from the form's control. Figure D illustrates this function, which is a simple but effective piece of code I used time and again in developing the custom Find Records dialog box.

The Check Expr pushbutton was originally designed to be a simple debugging tool. However, in an early use of FindBox in a live application, I neglected to remove it, only to find that my power users liked having it there. At a meeting with end users, I noticed the pushbutton's presence while they were showing me something else.

I casually mentioned that I'd mistakenly left it there and would remove it when I did some other work on the system, but a couple of the users vehemently protested. So, the Check Expr pushbutton has stayed put in most of the applications I've deployed. For your own applications, however, you might decide to eliminate this particular button. I still think it might scare some new users!

Putting FindBox to work

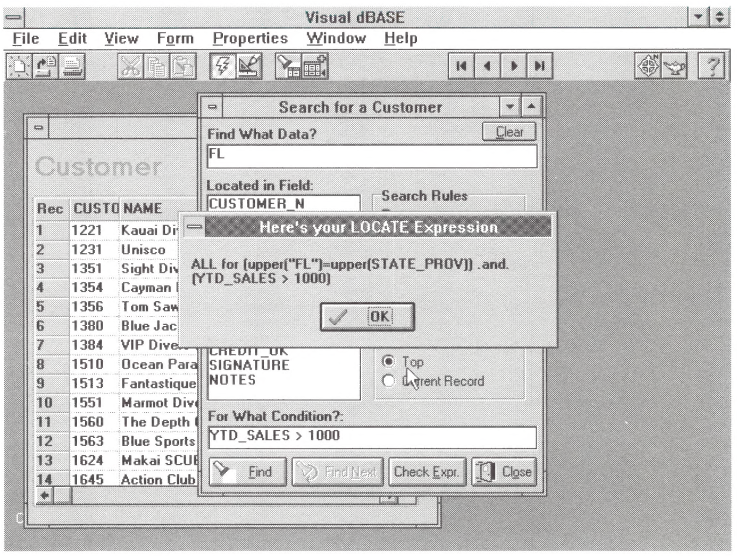
You can invoke FindBox with a simple DO FINDBOX.WFM command. Because it's a standard WFM file with just a little extra header code, it can be instantiated in all the ways you can create a form. By *instantiated*, we mean the way the DO, DEFINE, and NEW commands create a form from its CLASS definition.

By default, FindBox will use the currently active alias in the currently active session. The program takes up to two optional parameters: The first is an object referring to a form's View property, and the second is a character string title for the dialog box itself.

If you want to specify the title but not the view, simply pass an empty string as the first parameter. For example, in order to invoke FindBox from a Browse control's OnDoubleClick event handler, you might code the control like this:

```
DEFINE BROWSE BROWSE1 OF THIS;
   PROPERTY;
   OnLeftDblClick CLASS::BROWSE1_ONLEFTDBLCLICK,;
     Top 3,;
     Left 1,;
     Alias "CUSTOMER",;
     Height 16,;
     Width 56,;
     CUATab .T.

Procedure BROWSE1_OnLeftDblClick(flags, col, row)
      do findbox.wfm with ;
      "","Search for a Customer"
```

Figure D



Find What Data?
FL

Located in Field:
CUSTOMER_N

CREDIT_OK
SIGNATURE
NOTES

(● Top ○ Current Record)

For What Condition?:
YTD_SALES > 1000

Here's your LOCATE Expression

ALL for (upper("FL")=upper(STATE_PROV)) .and. (YTD_SALES > 1000)

OK

Rec	CUSTO	NAME
1	1221	Kauai Di
2	1231	Unisco
3	1351	Sight Div
4	1354	Cayman I
5	1356	Tom Saw
6	1380	Blue Jac
7	1384	VIP Dive
8	1510	Ocean Para
9	1513	Fantastique
10	1551	Marmot Div
11	1560	The Depth
12	1563	Blue Sports
13	1624	Makai SCU
14	1645	Action Club

The Check Expr button displays the LOCATE expression constructed from the values currently in use.

Shining the light

When attaching FindBox to a pushbutton, you should consider using the flashlight bitmaps in the default Visual dBASE resource file. Glyph #860 is the conventional bitmap for an enabled Find button, and #870 is its disabled counterpart. FindBox uses those bitmaps, too. By employing consistency in such details, you can develop a program that makes it easy for your end users to navigate your forms.

A work in progress

FindBox has served me well, and I hope you'll find it a nice addition to your programmer's toolbox. It seems every time I look over FindBox, I find some feature to add or improve. If you adapt FindBox in a unique way or enhance it to add new functionality, please share your work with me. In turn, I'll make new editions available via electronic media for other *Inside Visual dBASE* readers. ❖

Listing A: *FINDBOX.WFM*

```
* FINDBOX.WFM
* Replicates the good parts of the "Find Records" dialog box
* By Keith G. Chuvala
* CIS 71333,2654
* Email kgc@jinx.sckans.edu

parameter oView,cTitle
   local f
   f = NEW FindBoxForm()
   if type("cTitle") = "C"
      f.text = cTitle
   endif
   if type("oView") = "O"
      f.oView = oView
   endif
   f.Open()
return

** END HEADER - do not remove this line*
* Generated on 12/04/95
*
parameter bModal
local f
f = new FindBoxForm()
if (bModal)
   f.mdi = .F. && ensure not MDI
   f.ReadModal()
else
   f.Open()
endif
CLASS FindBoxForm OF FORM
   this.OnOpen = CLASS::SETUP
   this.Top = 0.2344
   this.Left = 3
   this.Text = "Locate Records"
   this.Height = 19.4111
   this.Width = 50
   this.OnSize = CLASS::SIZEPROC

   DEFINE RECTANGLE RECTANGLE2 OF THIS;
       PROPERTY;
         Top 3.5879,;
         Left 25,;
         Text "Search Rules",;
         ColorNormal "N/W",;
         Height 7.0586,;
         Width 22.666

   DEFINE RECTANGLE RECTANGLE1 OF THIS;
       PROPERTY;
         Top 11.1758,;
         Left 25.1641,;
         Text "Search From",;
         ColorNormal "N/W",;
         Height 3.4707,;
         Width 22.3359

   DEFINE ENTRYFIELD EF_FIND OF THIS;
       PROPERTY;
         Top 1.2939,;
         Left 0.6641,;
         Height 1.3525,;
         Width 48.8359,;
         Value "",;
         OnChange CLASS::EF_FIND_ONCHANGE,;
         MaxLength 100

   DEFINE LISTBOX LB_FIELD OF THIS;
       PROPERTY;
         Top 3.9404,;
         Left 0.8311,;
         ColorNormal "N/W*",;
         Height 10.7061,;
         Width 22.502,;
         ColorHighLight "W+/B",;
         ID 101

   DEFINE RADIOBUTTON RB_STANDARD OF THIS;
       PROPERTY;
         Top 4.4697,;
         Left 26,;
         Text "Standard",;
         ColorNormal "N/W",;
         Height 1.1182,;
         Width 15,;
         Value .T.,;
         FontBold .F.,;
         Group .T.

   DEFINE RADIOBUTTON RB_EXACT OF THIS;
       PROPERTY;
         Top 5.6465,;
         Left 26,;
         Text "Exact Length",;
         ColorNormal "N/W",;
         Height 1.1182,;
         Width 17.5,;
         Value .F.,;
         FontBold .F.,;
         Group .F.

   DEFINE RADIOBUTTON RB_ANYWHERE OF THIS;
       PROPERTY;
         Top 6.8232,;
         Left 26,;
         Text "Anywhere in Field",;
         ColorNormal "N/W",;
         Height 1.1172,;
         Width 19.833,;
         Value .F.,;
      FontBold .F.,;
      Group .F.

  DEFINE RADIOBUTTON RB_SOUNDEX OF THIS;
     PROPERTY;
       Top 8,;
       Left 26,;
       Text "Sounds Like",;
       ColorNormal "N/W",;
       Height 1.1172,;
       Width 20.5,;
       Value .F.,;
       FontBold .F.,;
       Group .F.

  DEFINE CHECKBOX CB_MATCH OF THIS;
     PROPERTY;
       Top 9.2939,;
       Left 26,;
       Text "Ignore Case",;
       ColorNormal "N/W",;
       Height 1.0586,;
       Width 19.333,;
       Value .F.,;
       FontBold .F.,;
       Group .T.

  DEFINE RADIOBUTTON RB_TOP OF THIS;
     PROPERTY;
       Top 12.0586,;
       Left 26.3311,;
       Text "Top",;
       ColorNormal "N/W",;
       Height 1.1172,;
       Width 15.002,;
       Value .T.,;
       FontBold .F.,;
       Group .T.

  DEFINE RADIOBUTTON RB_CURRENT OF THIS;
     PROPERTY;
       Top 13.0586,;
       Left 26.3311,;
       Text "Current Record",;
       ColorNormal "N/W",;
       Height 1.1172,;
       Width 18.6689,;
       Value .F.,;
       FontBold .F.,;
       Group .F.

  DEFINE ENTRYFIELD EF_FOR OF THIS;
     PROPERTY;
       Top 15.9404,;
       Left 0.8311,;
       Height 1.2939,;
       Width 48.502,;
       Value "",;
       OnChange CLASS::EF_FOR_ONCHANGE,;
       MaxLength 100

  DEFINE PUSHBUTTON PB_FIND OF THIS;
     PROPERTY;
       Top 17.5879,;
       Left 1,;
       Text "&Find",;
       ColorNormal "N/W",;
       Height 1.5293,;
       Width 11.5,;
       UpBitmap "RESOURCE #858",;
       DisabledBitmap "RESOURCE #868",;
       OnClick CLASS::PB_FIND_ONCLICK,;
       FontBold .F.,;
       Group .T.

  DEFINE PUSHBUTTON PB_FIND_NEXT OF THIS;
     PROPERTY;
       Default .T.,;
       Top 17.5879,;
       Left 13.165,;
       Text "Find &Next",;
       ColorNormal "N/W",;
       Height 1.5293,;
       Width 13.501,;
       UpBitmap "RESOURCE #860",;
       DisabledBitmap "RESOURCE #870",;
       OnClick CLASS::PB_FIND_NEXT_ONCLICK,;
       FontBold .F.,;
       Enabled .F.,;
       Group .T.

  DEFINE PUSHBUTTON PB_CHECK OF THIS;
     PROPERTY;
       Top 17.5879,;
       Left 27.332,;
       Text "Check &Expr.",;
       ColorNormal "N/W",;
       Height 1.5293,;
       Width 11.501,;
       OnClick CLASS::PB_CHECK_ONCLICK,;
       FontBold .F.,;
       Group .T.

  DEFINE PUSHBUTTON PB_CLOSE OF THIS;
     PROPERTY;
       Top 17.5879,;
       Left 39.5,;
       Text "Cl&ose",;
       ColorNormal "N/W",;
       Height 1.5293,;
       Width 9.666,;
       UpBitmap "RESOURCE #1005",;
       OnClick {; form.close()},;
       FontBold .F.,;
       Group .T.

  DEFINE TEXT TEXT1 OF THIS;
     PROPERTY;
       Top 0.4111,;
       Left 0.833,;
       Text "Find What Data?",;
       ColorNormal "N/W",;
       Height 0.8232,;
       Width 17.333

  DEFINE TEXT TEXT2 OF THIS;
     PROPERTY;
       Top 3.0586,;
       Left 0.6641,;
       Text "Located in Field:",;
       ColorNormal "N/W",;
       Height 0.7646,;
       Width 17.3359

  DEFINE TEXT TEXT3 OF THIS;
     PROPERTY;
       Top 15.0586,;
       Left 0.8311,;
       Text "For What Condition?:",;
       ColorNormal "N/W",;
       Height 0.7646,;
       Width 25.002

  DEFINE PUSHBUTTON PB_NEWFILE OF THIS;
     PROPERTY;
       Top 2.8818,;
       Left 19.5,;
       Text "",;
       ColorNormal "N/W",;
      Height 0.9414,;
      Width 3.833,;
      UpBitmap "RESOURCE #144",;
      OnClick CLASS::PB_NEWFILE_ONCLICK,;
      Group .T.

DEFINE PUSHBUTTON PB_CLEAR OF THIS;
    PROPERTY;
      Top 0.1758,;
      Left 41.1641,;
      Text "&Clear",;
      ColorNormal "N/W",;
      Height 1,;
      Width 8.002,;
      OnClick CLASS::PB_CLEAR_ONCLICK,;
      FontBold .F.,;
      SpeedBar .T.,;
      Group .T.

Procedure Setup
   if .not. isblank(alias())
      form.lb_field.datasource = "structure"
      form.pb_newfile.visible = .f.
   else
      form.pb_newfile.visible = .t.
   endif
   form.oheight = form.height
   form.owidth = form.width
return

Procedure sizeproc
   if form.height <> form.oheight
      form.height =  form.oheight
   endif
   if form.width <> form.owidth
      form.width = form.owidth
   endif
   if form.windowstate = 2
      form.windowstate = 0
   endif
return

Procedure pb_newfile_OnClick
   cFile = getfile("*.dbf;*.qbe;*.db")
   if .not. isblank(cFile)
      if type("form.oView") = "O"
         form.oView.view = cFile
      else
         form.view = cFile
      endif
      form.lb_field.datasource = "structure"
   endif
return

Procedure pb_find_onclick
   if class::Search(.t.)
      form.pb_find_next.enabled = .t.
   endif
return

Procedure pb_check_onclick
   class::Search(.f.)
return

Procedure Search
   parameter lFind
   local nErr,nRn,lFound
   lFound = .f.
   if isblank(alias())
      class::pb_newfile_Onclick()
   else
      cFind1 = trim(form.ef_find.value)
      cFind2 = trim(form.lb_field.value)
      cFind3 = trim(form.ef_for.value)
```

```
      cAddfor = ""
      cExp = ""

      cFor = ""
      if isblank(cFind1) .and. isblank(cFind3)
       msgbox("You need to specify something to search for!",;
       "Can't Search Yet!")
         form.ef_find.setfocus()
         return .f.
      endif

      if isblank(cFind2) .and. isblank(cFind3)
         msgbox("You need to select a field to search in!",;
         "Can't Search Yet!")
         form.lb_field.setfocus()
         return .f.
      endif

      cScope = iif(form.rb_top.value,"ALL ",;
               iif(form.rb_current.value,"REST ",""))

      cType =  iif(form.rb_standard.value,"=",;
               iif(form.rb_exact.value,"==",;
               iif(form.rb_anywhere.value,"$","")))

      cFtype = type(cFind2)
      do case
         case cFtype = "C"
            if .not. left(cFind1,1) $ ['"]+"["
               cFind1 = '"'+cFind1+'"'
               if form.cb_match.value
                  cFind1 = "upper("+cFind1+")"
                  cFind2 = "upper("+cFind2+")"
               endif
            endif
         case cFtype = "D"
            if left(cFind1,1) <> "{"
               cFind1 = "{"+cFind1+"}"
            endif
      endcase

      if .not. isblank(cFind1)
         if .not. isblank(cType)
            cFor = "for ("+cFind1+cType+cFind2+")"
         else
            cFor = ;
            "for (soundex("+cFind1+") == soundex("+cFind2+"))"
         endif
      endif

      cAddFor = iif(.not. isblank(cFind3),;
                    iif(.not. isblank(cFor)," .and. ;
                         (","for (")+cFind3+")", "")
      cExp = cScope+cFor+cAddFor

      #ifdef DEBUG
         lFind = .f.
      #endif
      if .not. lFind
         msgbox(cExp,"Here's your LOCATE Expression")
      else
         nErr = 0
         on error nErr = 1
         nRn = recno()
         locate &cExp
         on error
         if nErr = 1
            msgbox(message(),"Invalid search condition")
            if isblank(cFind3)
               form.ef_find.setfocus()
            else
               form.ef_for.setfocus()
            endif
         else
         if .not. found()
            go nRn
            lFound = .f.
            msgbox("No records found. ;
            Try 'Check Expr.', then try again","No Match!")
         else
            lFound = .t.
         endif
      endif
   endif
endif
return lFound

Procedure PB_Clear_OnClick
   form.ef_for.value = ""
   form.lb_field.value = ""
   form.lb_field.cursel = 0
   form.ef_find.value = ""
   form.ef_find.setfocus()
return
```

```
   Procedure EF_FIND_OnChange
      if .not. isblank(this.value)
         form.pb_find.enabled = .t.
         form.pb_find_next.enabled = .f.
      endif

   Procedure EF_FOR_OnChange
      if .not. isblank(this.value)
         form.pb_find.enabled = .t.
         form.pb_find_next.enabled = .f.
      endif

   Procedure PB_FIND_NEXT_OnClick
      continue
      if .not. found()
         this.enabled = .f.
         form.pb_find.setfocus()
      endif

ENDCLASS
```

Using SUPER

One of Visual dBASE's greatest strengths is its object-oriented language. Object-oriented programming, or OOP, produces applications that have more cohesive designs and are easier to maintain. The larger and more complex the application, the more apparent the benefits of OOP become.

In this article, we'll discuss one of OOP's fundamental features—*polymorphism*—which means "many forms." In a number of computer languages, including dBASE, you usually extend that to mean "one name, many forms," or "one function name, different functions." We'll also present sample code you can run to demonstrate how polymorphism affects application development.

Changing behaviors

It's easy to use polymorphism to change the behavior of any function in dBASE. **Listing A** demonstrates how you can alter a Form object's Move( ) function to display the form's position on the screen each time it moves.

If you run this little piece of code, you'll notice one important side effect: The form no longer moves! When you write a function that has the same name as an existing function, the new function overrides the original—an example of polymorphism in action. In cases like this one, however, you want to add new functionality *and* retain the behavior of the function as it behaved in the base class.

Visual dBASE has a special keyword that allows you to call the function inherited from the class from which your new class is derived: SUPER. Take a look at how we use SUPER in **Listing B**. The class is almost identical to our earlier effort, with one addition: SUPER::Move( ).

Form Tip

Listing A: *Using polymorphism to change the behavior of FORM::Move( )*

```
CLASS MoveForm of Form
   Procedure Move
      parameter nTop,nLeft,nWidth,nHeight
      local cText,nCoordPos
      cText = Form.Text
      nCoordPos = at(">>",cText)
      if nCoordPos = 0
         Form.Text = "<<"+ltrim(str(Form.Top)) + "," + ;
                     ltrim(str(Form.left)) + ">>   " + cText
      else
         Form.Text = "<<" + ltrim(str(Form.Top)) + "," + ;
                     ltrim(str(Form.Left)) + ">>" + ;
                     substr(cText,nCoordPos+2)
      endif
ENDCLASS
```

Listing B: *Calling SUPER::Move( )*

```
CLASS MoveForm of Form
   Procedure Move
      parameter nTop,nLeft,nWidth,nHeight
      local cText,nCoordPos

      SUPER::Move(nTop,nLeft,nWidth,nHeight)

      cText = Form.Text
      nCoordPos = at(">>",cText)
      if nCoordPos = 0
         Form.Text = "<<"+ltrim(str(Form.Top)) + "," + ;
                     ltrim(str(Form.left)) + ">>   " + cText
      else
         Form.Text = "<<" + ltrim(str(Form.Top)) + "," + ;
                     ltrim(str(Form.Left)) + ">>" + ;
                     substr(cText,nCoordPos+2)
      endif

ENDCLASS
```

This instruction tells dBASE to run the Move( ) function as it existed in the base class. So, the new form class not only moves like the original base class form, it also includes our custom functionality of adding the coordinates to the title bar each time you move the form.

Opening up

One of the most popular uses of SUPER is to extend the functionality of the Open( ) function. Open( ) is a very important function, since you can't display your form without it. In many cases, however, you'll want to set properties in the form before it becomes visible on the screen.

In order to accomplish that objective, you can create a new Open( ) function that does all the setup work first, then calls SUPER::Open( ) as the last step. This approach ensures that the form will open onscreen as expected.

You could accomplish the same effect inside an event handler for the OnOpen event. However, in event-handler code, the form opens first, then the code executes. For centering or otherwise altering the appearance of a form, this behavior means the user sees the form moving and changing as it takes shape. Even judicious manipulation of the Form.Visible property to hide such maneuvers often results in a form that appears and then immediately disappears, only to reappear once the code has executed. Fortunately, you can avoid that kind of screen flicker.

The code in **Listing C** shows how to center a form onscreen before it opens and becomes visible. Note

Listing C: *A form that self-centers any time you open it*

```
Class CenterForm of FORM
   Procedure OPEN
      do appunits
      form.top  = (_app.height / 2) - (form.height / 2)
      form.left = (_app.width  / 2) - (form.width  / 2)
   SUPER::Open()
EndClass

PROCEDURE AppUnits
*------------------------------------
*- Programmer...: Ken Chan (Zak) (CIS: 72662,1305)
*- Date.........: 07/06/1994
*- Notes........: Determine the application's height/width
*-                for use in centering dialog boxes.
*- Written for..: dBASE 5.0 for Windows
*- Rev. History.: 04/22/1994 - Original
*-                07/06/1994 - Modified due to changes in
*-                dBASE/Win -
*-                According to Zak, this should _always_
*-                work, even if the internal units change.
*- Calls........: ByteVal()      && below
*- Called by....: Class: Dialog
*- Usage........: Do GetAppUnits
*- Example......: Do GetAppUnits
*- Returns......: None
*- Parameters...: None
*------------------------------------
  if type( "_app.Width" ) = "U"
     extern CINT GetSystemMetrics( CINT ) USER.EXE
     #define SM_CXSCREEN 0
     #define SM_CYSCREEN 1
     _app.XPixels = GetSystemMetrics( SM_CXSCREEN )
     _app.YPixels = GetSystemMetrics( SM_CYSCREEN )
     #define FORM_HEIGHT 4
     #define FORM_WIDTH  20
     define form fConfig property ;
        top -100, Text"AutoConfig", ;
        Height   FORM_HEIGHT, ;
        Width    FORM_WIDTH, ;
        MDI .F., Sizeable .F., SysMenu  .F.
     fConfig.open()
     extern    CVOID   GetWindowRect ( CHANDLE, CPTR ) USER.EXE
     cRect = space( 8 )
     GetWindowRect( fConfig.hWnd, cRect )
     nWinHeight = ByteVal( cRect, 7, 2 ) - ByteVal( cRect, 3,  2 )
     extern CVOID GetClientRect( CHANDLE, CPTR ) USER.EXE
     cRect = space( 8 )
     GetClientRect( fConfig.hWnd, cRect )
     nClientWidth  = ByteVal( cRect, 5,  2 )
     nClientHeight = ByteVal( cRect, 7, 2 )
     fConfig.close()
     _app.XUnits  = nClientWidth  / FORM_WIDTH
     _app.YUnits  = nClientHeight / FORM_HEIGHT
     _app.THeight = ( nWinHeight - nClientHeight ) / _app.YUnits
     extern CLONG GetDialogBaseUnits( CVOID ) USER.EXE
     #define WORD_SIZE 65536
     local nDouble
     nDouble = GetDialogBaseUnits()
     _app.XDlgUnits  = mod( nDouble, WORD_SIZE )
     _app.YDlgUnits  = int( nDouble / WORD_SIZE )
     _app.Width   = _app.XPixels / _app.XUnits
     _app.Height  = _app.YPixels / _app.YUnits
  endif
RETURN
*- EoP: GetAppUnits

FUNCTION ByteVal(cArg,nPos,nLen)
*--------------------------------------------
*- Programmer...: Ken Chan (Zak) (CIS: 72662,1305)
*- Date.........: 07/06/1994
*------------------------------------------
     if pcount() < 3
        nLen = 1
     endif
     local nRet, nPtr
     nRet = 0
     for nPtr = nPos to nPos + nLen - 1
        nRet = nRet + ;
        asc( substr( cArg, nPtr, 1 )) * 256 ^ ( nPtr - nPos )
     endfor
RETURN nRet
*- EoF: ByteVal()
```

that all the Windows API calls necessary to calculate the proper coordinates occur before the form opens, so the user is unaware that anything is happening. From the user's perspective, the form simply opens in the right place.

Conclusion

Some OOP concepts and techniques might seem esoteric or hard to use. However, SUPER is easy to use and gives you the ability to extend Visual dBASE's functions to even more powerful and flexible applications. ❖

Thanks to Zak!

Our thanks go to Kenneth "Zak" Chan—a long-time dBASE enthusiast and TeamB member who now works for Borland—for contributing the AppUnits and ByteVal code in Listing C in the accompanying article. (Those routines figure the width and height of the screen in dBASE-friendly units.) AppUnits is part of the dBASE Users Function Library Project (dUFLP) library, available on CompuServe and on various BBSs and Internet FTP sites.

Avoiding Query Designer "gotchas"

Visual dBASE's Query Designer is a powerful tool that allows you to graphically design the relationships between open tables, specify criteria for record-matching conditions, create calculated fields, select a sort or index order, and more. It also has a few eccentricities you'll want to avoid altogether. In this article, we'll discuss two unexpected "gotchas" that sometimes pop up when you're using the Query Designer, and we'll show you how to work around or avoid them.

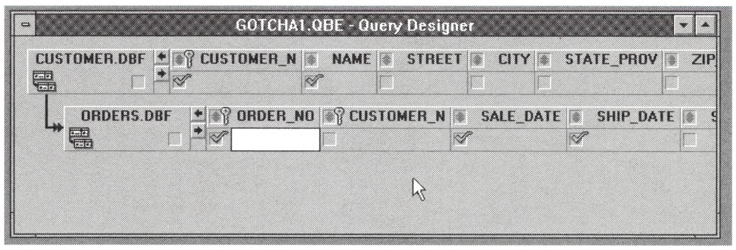
Know EXACTly what you're doing!

Our first Query Designer gotcha is insidious, although the symptom is easy to spot. You create a query with a parent table and a child table. Then, you connect the two tables with a relation between similar fields. You set the order of your fields and the desired sorting order, etc. After looking over your beautiful query, you decide it's perfect and press the Run button on the SpeedBar.

To your dismay, you see that the query selected no records, and you break out in a cold sweat. You check, recheck, and triple-check everything. Finally, you press the Run button again in the desperate hope that the first time dBASE just wasn't awake or paying attention. However, you see the *No records selected* message again and run from the room screaming and pulling out your hair by the fistful. Before you're entirely bald, however, you realize that the problem may have been that you didn't tell Visual dBASE *exactly* what you wanted. Let's work through an example to see what's going on.

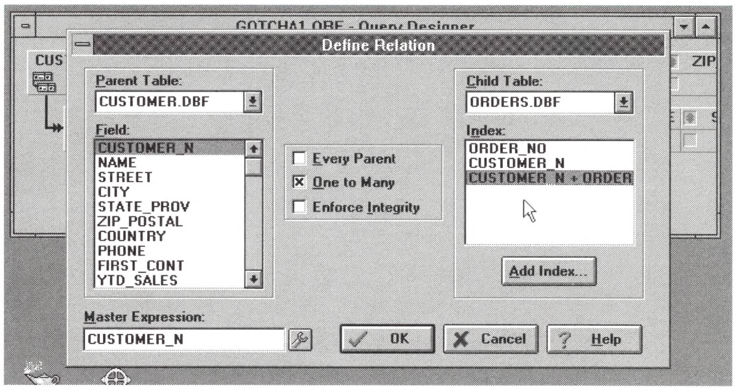
Figure A shows a query we created using the sample tables CUSTOMER.DBF and ORDERS.DBF from the Visual dBASE SAMPLES directory. Figure B shows how we established the relationship between the two tables. It makes sense to use the

Figure A



Here's our seemingly innocent query comprised of the Visual dBASE sample tables, CUSTOMERS.DBF and ORDERS.DBF.

Figure B



Parent Table: CUSTOMER.DBF

Field: CUSTOMER_N, NAME, STREET, CITY, STATE_PROV, ZIP_POSTAL, COUNTRY, PHONE, FIRST_CONT, YTD_SALES

☐ Every Parent
☒ One to Many
☐ Enforce Integrity

Child Table: ORDERS.DBF

Index: ORDER_NO, CUSTOMER_N, CUSTOMER_N + ORDER

Add Index...

Master Expression: CUSTOMER_N

OK Cancel Help

The relationship between the two tables is based on CUSTOMER_N. Or is it?

CUSTORD index tag in the child table, since that tag uses the customer number field (CUSTOMER_N) plus the order number field (ORDER_N). This order will produce a query where the orders are grouped by customer number, then listed by order number within the child table, right? Figure C answers the question in nose-tweaking fashion.

Beware of compound tags

If you're an experienced dBASE Query Designer, you may have already figured out this dilemma. The problem is that the CUSTORD tag isn't the same length as the CUSTOMER_N field in the parent table. Therefore, no matches can be made.

On the other hand, if you work much from the dBASE dot prompt, you'd just *swear* you've written relationships like this one that have worked just fine. The problem is that those relationships work, but only when the SET EXACT parameter is OFF.

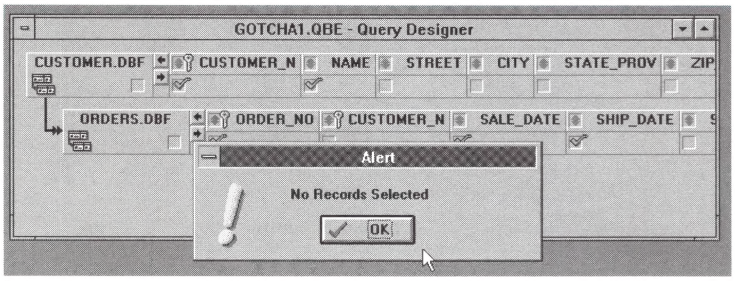
When EXACT is ON, any comparison between character fields (including those used in establishing a relationship) will fail or return a false (.F.) result if the two fields are different lengths—even if the data they contain are otherwise identical. This behavior is planned and is actually a useful safety net built into the dBASE language.

However, if that behavior is cramping your style, you can take advantage of the fact that Query Designer is a two-way tool—used both to figure out the problem and to provide a solution. Listing A shows the code that the Query Designer created from our definition. Note that the second line of the code reads SET EXACT ON. Any time you double-click on this query's icon, invoke it with a SET VIEW TO command, or set it up as the View property of a Form control or Data control, Visual dBASE will execute that line of code, resulting in the persnickety only-exact-matches-welcome-here mode. The Query Designer always removes this line of code. In this case, we don't want to include that line of code, so we'll edit the query as code and fix it.

To edit a query in a code window, right-click on the query's icon in Navigator and select Edit As Program. Make the change as illustrated in Figure D, changing SET EXACT ON to SET EXACT OFF. Save the query by pressing [Ctrl]W, selecting File | Save from the menu bar, or doing whatever else your favorite save-and-exit method might happen to be.

Now let's test the change. Double-click on the query's icon in Navigator. Ahhhh, much better! Figure E shows the results. We've lined up all customers with their correct orders, and listed each in ascending customer number order.

Figure C



No Records Selected

The error message tells the tale of a query definition gone awry.

A caveat and a request

The fix we describe in this article works beautifully, but there's one warning to bear in mind. Because the Query Designer insists on streaming out the command as SET EXACT ON, you'll have to remember to patch the query code as we've demonstrated here each time you modify it with the Query Designer.

If you're on CompuServe, please visit the Visual dBASE forum (GO VDBASE) and post a message to the Wish List section asking that SET EXACT be made a switchable setting in future updates to the product. This simple enhancement to the program would eliminate this gotcha altogether and make the Query Designer even more useful than it is now!

Listing A: *GOTCHA1.QBE*

```
* Visual dBASE .QBE file 9
CLOSE DATABASES
SET EXACT ON
SELECT 1
USE \VDB\SAMPLES\CUSTOMER.DBF
SELECT 2
USE \VDB\SAMPLES\ORDERS.DBF ORDER TAG CUSTORD
SELECT 1
SET RELATION TO CUSTOMER_N INTO ORDERS CONSTRAIN
SET SKIP TO ORDERS
SET FILTER TO FOUND(2)
SET FIELDS TO CUSTOMER_N, NAME
SELECT 2
SET FIELDS TO SALE_DATE, SHIP_DATE, AMT_PAID
SELECT 1
GO TOP
```

The Path trap

The second Query Designer gotcha is a problem that primarily affects deployed applications. Go back and review Listing A, paying careful attention to the filenames in the two USE statements. Notice that the Query Designer streamed out the full path to each table. This behavior poses no problem for a standalone system or for a networked copy of dBASE in an environment where everyone's drive letters are mapped consistently. However, it can be a big problem if you use the compiler and Application Deployer to distribute your Visual dBASE applications.

The Setup program created by the Application Deployer allows users to specify the destination directory at the time of installation, as shown in Figure F. The directory names—and in many cases, even the drive letters—can differ in the deployed environment from the way they appear on your development system, leading to aggravating *Error: File does not exist* messages.

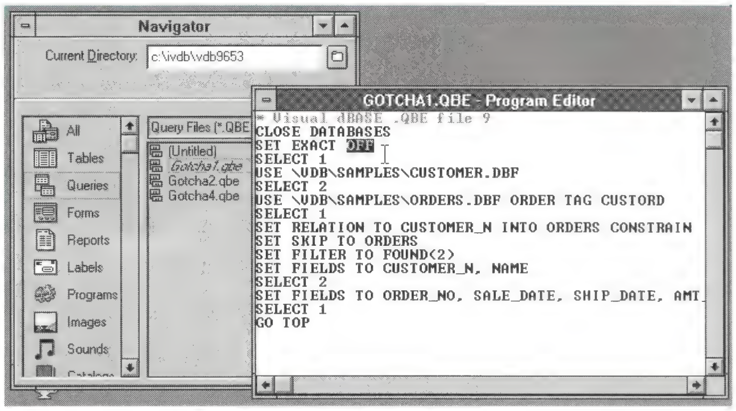
Two-way tools to the rescue again

We solved the first gotcha by using the two-way nature of the Query Designer—editing the code it produces so we could make the fix directly to the query code. We can address this second problem in the same way, by editing out all the literal drive letters and paths. Listing B, on the next page, shows the changed query code. This example assumes that all tables will be in the same directory as the compiled EXE file. If your application is designed to store tables in subdirectories, relative references to those directories will work too. Listing C, also on the next page, demonstrates this.

BDE aliases: a better fix!

Another workaround to the Path trap problem is to use Borland Database Engine (BDE) aliases. A BDE alias is different from a dBASE alias as we've come to know the term. Instead, a BDE alias describes the location of a database—which might be a SQL server, an ODBC driver, or just a directory somewhere on the system. You can use the BDE Config Utility in the

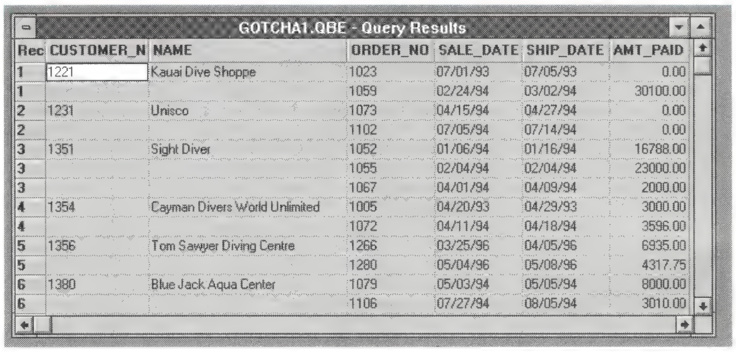
Figure D



```
* Visual dBASE .QBE file 9
CLOSE DATABASES
SET EXACT OFF
SELECT 1
USE \VDB\SAMPLES\CUSTOMER.DBF
SELECT 2
USE \VDB\SAMPLES\ORDERS.DBF ORDER TAG CUSTORD
SELECT 1
SET RELATION TO CUSTOMER_N INTO ORDERS CONSTRAIN
SET SKIP TO ORDERS
SET FILTER TO FOUND(2)
SET FIELDS TO CUSTOMER_N, NAME
SELECT 2
SET FIELDS TO ORDER_NO, SALE_DATE, SHIP_DATE, AMT_
SELECT 1
GO TOP
```

You can use the Edit As Program *function to fix the query.*

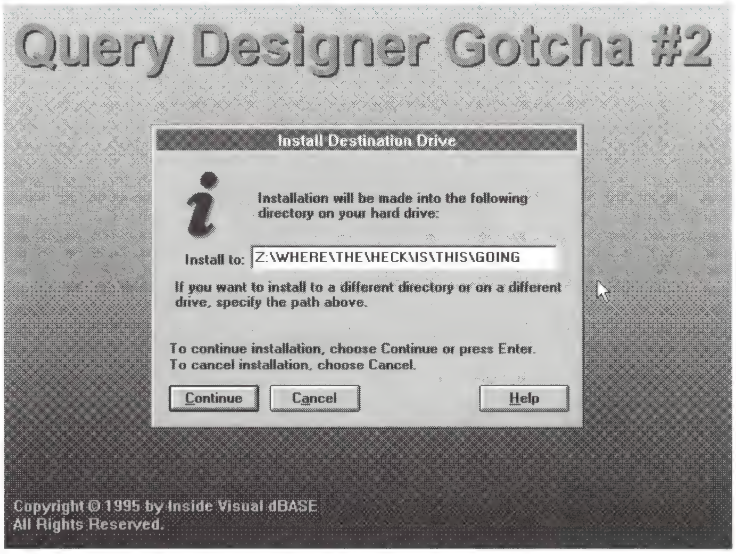
Figure E



Rec	CUSTOMER_N	NAME	ORDER_NO	SALE_DATE	SHIP_DATE	AMT_PAID
1	1221	Kauai Dive Shoppe	1023	07/01/93	07/05/93	0.00
1			1059	02/24/94	03/02/94	30100.00
2	1231	Unisco	1073	04/15/94	04/27/94	0.00
2			1102	07/05/94	07/14/94	0.00
3	1351	Sight Diver	1052	01/06/94	01/16/94	16788.00
3			1055	02/04/94	02/04/94	23000.00
3			1067	04/01/94	04/09/94	2000.00
4	1354	Cayman Divers World Unlimited	1005	04/20/93	04/29/93	3000.00
4			1072	04/11/94	04/18/94	3596.00
5	1356	Tom Sawyer Diving Centre	1266	03/25/96	04/05/96	6935.00
5			1280	05/04/96	05/08/96	4317.75
6	1380	Blue Jack Aqua Center	1079	05/03/94	05/05/94	8000.00
6			1106	07/27/94	08/05/94	3010.00

Customers with orders are made to order, so to speak.

Figure F



Installation will be made into the following directory on your hard drive:

Install to: Z:\WHERE\THE\HECK\IS\THIS\GOING

If you want to install to a different directory or on a different drive, specify the path above.

To continue installation, choose Continue or press Enter.
To cancel installation, choose Cancel.

Copyright © 1995 by Inside Visual dBASE
All Rights Reserved.

The user can specify any valid destination path for a deployed application.

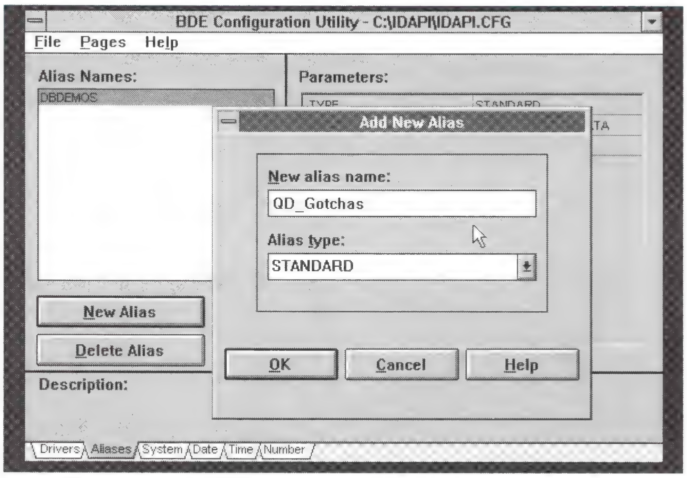
Listing B: *GOTCHA2.QBE*

```
* Visual dBASE .QBE file 9
CLOSE DATABASES
SET EXACT ON
SELECT 1
USE CUSTOMER.DBF
*NOTE! Drive and path references have been removed!
SELECT 2
USE ORDERS.DBF ORDER TAG CUSTORD
** NOTE! Drive and path references have been removed!
SELECT 1
SET RELATION TO CUSTOMER_N INTO ORDERS CONSTRAIN
SET SKIP TO ORDERS
SET FILTER TO FOUND(2)
SET FIELDS TO CUSTOMER_N, NAME
SELECT 2
SET FIELDS TO SALE_DATE, SHIP_DATE, AMT_PAID
SELECT 1
GO TOP
```

Listing C: *GOTCHA3.QBE*

```
* Visual dBASE .QBE file 9
CLOSE DATABASES
SET EXACT ON
SELECT 1
USE DATA\CUSTOMER.DBF
** NOTE! Changed to use relative pathnames!
SELECT 2
USE ORDERS.DBF ORDER TAG CUSTORD
* NOTE! Changed to use relative pathnames!
SELECT 1
SET RELATION TO CUSTOMER_N INTO ORDERS CONSTRAIN
SET SKIP TO ORDERS
SET FILTER TO FOUND(2)
SET FIELDS TO CUSTOMER_N, NAME
SELECT 2
SET FIELDS TO SALE_DATE, SHIP_DATE, AMT_PAID
SELECT 1
GO TOP
```

Figure G



New alias name: QD_Gotchas

Alias type: STANDARD

You set up a BDE alias in the BDE Configuration Utility dialog box.

Visual dBASE Program Manager Group to create an alias.

To do so, just select the Alias tab, press the New Alias button, and then type a name for your new alias, as shown in Figure G. After you click OK, enter the name of the directory that contains the tables for your application in the Path field. Then, select Save from the File menu, and exit the BDE Configuration Utility.

After you make this change, when you return to Visual dBASE, you'll be able to select the Tables From Directory or Tables From Database options in Navigator. To see how this works, select Tables From Database, and create a new query by selecting one of the tables from the file list. Create a simple query and save it. Back in Navigator, right-click on your new query's icon, and select Edit As A Program. Now you can see the difference an alias makes.

Listing D shows a simple alias we created with one of our sample tables. Note the different syntax in the USE command for the table—no drive letters and no directory paths. The database alias name is delimited with colons, a practice which might be unfamiliar to long-time dBASE for DOS users. Colon delimiters were added to Visual dBASE to deal with SQL and ODBC data sources that contain spaces and other characters that are illegal in traditional dBASE syntax.

An advantage of using BDE aliases is that the Query Designer will never insert literal drive letters and directory paths. Stripping unwanted literal references from the query code works fine, but must be done each time you use the Query Designer to modify the query. On the other hand, if you use aliases, you've got a permanent solution to this particular gotcha!

Listing D: *GOTCHA4.QBE*

```
* Visual dBASE .QBE file 9
CLOSE DATABASES
SET EXACT ON
SELECT 1
OPEN DATABASE QD_GOTCHA
USE :QD_GOTCHA:SW.DBF
SET FIELDS TO SERIALNO, PROGRAM, REGDATE
GO TOP
```

More about aliases

Besides the benefits we've described, there are some important issues beyond query design that arise from using BDE aliases in applications. If you'd like to read more about the BDE, aliases, and related tricks and traps, let us hear from you. If there's sufficient interest in the topic, we'll present articles on this topic again.

Winding up

The Query Designer isn't perfect. However, in this article, we showed you how you can use Visual dBASE's two-way tools to adjust and optimize the code that the Query Designer produces in order to suit your needs. In future issues, we'll continue to look at the Query Designer's quirks, as well as its more powerful features. ❖

Directing the eye with color

Design Tip

A busy form can be tough on the eyes of data entry personnel. The default Windows cursor, which is a mere flashing vertical line, doesn't help matters. In this article, we'll show you how you can alleviate eye strain and reduce errors by using colors that help the user easily identify which control or field on the screen has focus.

The ColorHighlight property

Visual dBASE makes it easy to customize the color scheme for many controls. The ColorNormal property specifies the control's color while it doesn't have focus, and the ColorHighlight property specifies the control's color when it does have focus. Figure A shows a form that makes use of these two properties. Only the field with focus is brightly colored, which in this example is WindowText/Window. All other windows are kept dim (WindowText/BtnFace).

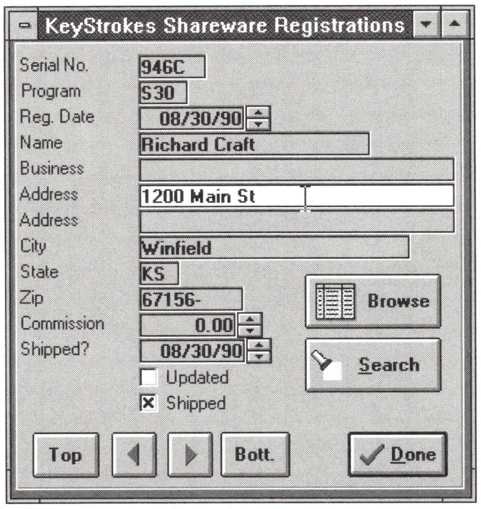
Borland added ColorHighlight in the transition from dBASE for Windows 5.0 to Visual dBASE 5.5. Not all controls were given equal treatment, however. Check boxes, radio buttons, combo boxes, and pushbuttons don't have the ColorHighlight property, even though they're also good candidates for custom color treatment.

Coloring your own

Fortunately, all controls have the ColorNormal property, as well as event handlers for OnGotFocus and OnLostFocus. To add custom coloring to a check box, you can simply change the ColorNormal property each time one of these two events occurs. Listing A demonstrates how to accomplish this effect using simple code blocks in the check box definition.

If you use this technique once, chances are good that you'll use it many times. To avoid typing in similar code blocks for each control on your form, you can save

Figure A


KeyStrokes Shareware Registrations

Serial No.	946C
Program	S30
Reg. Date	08/30/90
Name	Richard Craft
Business	
Address	1200 Main St
Address	
City	Winfield
State	KS
Zip	67156-
Commission	0.00
Shipped?	08/30/90

☐ Updated
☒ Shipped

You can make a form easier to use by brightening the field with focus and dimming the others.

Listing A: *Adding color changes to a check box control*

```
DEFINE CHECKBOX CHECKBOX1 OF THIS;
       PROPERTY;
         OnGotFocus {; this.ColorNormal = "r+/BtnFace"},;
         OnLostFocus {; this.ColorNormal = "BtnText/BtnFace"},;
         Group .T.,;
         Text "This will turn red!",;
         Height 1.25,;
         Width 20,;
         Left 5,;
         Top 5
```

time by creating a set of custom classes that do the work for you. Listing B contains the code for a custom control that you can add to the Control Palette's Custom tab in Form Designer.

You can easily construct similar custom controls for radio button, combo box, and pushbutton controls. Figure B shows our form with shiny new check boxes that are highlighted when active, making for a more user-friendly data entry environment.

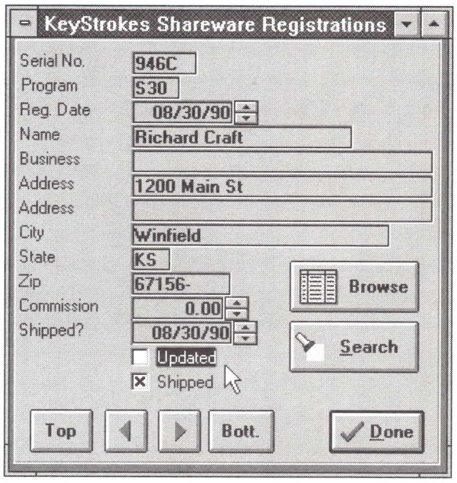
Listing B: *Creating a chameleon check box custom control*

```
CLASS colorbox(FormObj,Name) OF CHECKBOX(FormObj,Name) Custom
   this.OnGotFocus = CLASS::Red
   this.OnLostFocus = CLASS::Normal
   this.Text = "This will turn red!!!"

   Procedure Red
      this.colornormal = "r+/BtnFace"

   Procedure Normal
      this.colornormal = "BtnText/BtnFace"

ENDCLASS
```

Conclusion

Judicious use of color in your forms can make them easier to use and more pleasing to the eye. Fortunately, creating custom color schemes doesn't take a lot of effort on your part, and the folks who use your programs will notice and appreciate the difference! ❖

Figure B



KeyStrokes Shareware Registrations

Serial No.	946C
Program	S30
Reg. Date	08/30/90
Name	Richard Craft
Business	
Address	1200 Main St
Address	
City	Winfield
State	KS
Zip	67156-
Commission	0.00
Shipped?	08/30/90

☐ Updated
☒ Shipped

Changing the color of the active check box can make data entry forms easier to use.

MegaDittos from dBASE!

We based this article on a question submitted by JoAnn Nowatzke, who hails from Ottawa, Kansas.

In dBASE for DOS, when you're working in BROWSE, EDIT, or APPEND modes, you can copy a field value from the previous record into the current record by pressing Ditto ([Shift][F8]). If you upgraded to Visual dBASE from a previous release, you've probably discovered that the Ditto function was omitted from the Windows version. In this article, we'll look at two approaches to replicating this function in the Windows environment by creating your own Ditto key in Visual dBASE.

The Ditto function

When you consider how to implement a Ditto keypress in Visual dBASE, two possible solutions emerge right away: the STORE AUTOMEM command and the Windows Clipboard. Let's discuss each of these approaches in detail.

The STORE AUTOMEM approach

The first solution involves the STORE AUTOMEM command. With this approach, you simply capture field values with STORE AUTOMEM in the OnNavigate event handler, then REPLACE those values into the current record when the operator presses [Shift][F8].

Although this approach works, it's somewhat unwieldy. Specifically, it's easy enough to capture field values with STORE AUTOMEM in the OnNavigate event handler. However, if the user happens to be using the [⬆] key to navigate, keeping track of which record appears above the current row becomes a problem.

Since the OnNavigate event fires after the record pointer has moved, you must write a routine that updates the memory variables one step behind the record pointer, based on the order in which the records appear and on the keys the operator uses to navigate between records. If you're comfortable programming the logic for this type of routine, you can use the STORE AUTOMEM approach to simulate the Ditto function. However, you may find that it's easier to use the Windows Clipboard approach.

The Windows Clipboard approach

The simplest and most elegant solution to replacing the Ditto function is to think of that function as a copy-and-paste operation from one record to another. Although Visual dBASE makes it easy to navigate from record to record, you run into difficulty coding the copy-and-paste parts of the process.

If you've worked much with forms, you already know that the dBASE language has been extended to support the EditCopyMenu and EditPasteMenu instructions. However, these commands are designed to be attached to menu bar objects, so they're not useful for forms that don't have menus or for the stand-alone BROWSE command.

Windows provides a number of API functions that can get the job done, but Visual dBASE already knows about those functions. What we need is an easy way to use them.

Windows Clipboard programming is a bit scary if you're not familiar with the nuts and bolts of the API functions that manipulate it. Fortunately, we can use the dBASE KEYBOARD command to do all the difficult parts for us. As you probably know, the standard keypress for copying a selection to the Clipboard is [Ctrl]C, and the keypress for pasting data into an application is [Ctrl]V. To copy the value from the field in the preceding record, our routine needs to:

- navigate up one record
- copy the selection to the Clipboard
- navigate down one record
- paste the Clipboard's contents into the original field

Now, let's go over each step in detail.

Using the KEYBOARD( ) approach

Sometimes the brute force method of entering values in a field works best, and that's the approach we'll use. Specifically, instead of wrestling with complex routines, we can use the KEYBOARD( ) function to easily perform the copy-and-paste operation. KEYBOARD( ) stuffs keystrokes into the computer's keyboard buffer as if they'd been typed from the keyboard.

To facilitate special keystrokes such as function keys, control keys, and the like,

INSIDE VISUAL dBASE™

Inside Visual dBASE (ISSN 1084-1970) is published monthly by The Cobb Group.

Staff:	Contributing Editor-in-Chief	Keith G. Chuvala
	Associate Editors-in-Chief	Jeff E. Davis
		Tiffany M. Taylor
	Publications Coordinator	Maureen Spencer
	Editors	Laura Merrill
		Joan McKim
		Elisabeth Pehlke
	Production Artist	Alison Schwarz
	Product Group Manager	Mike Stephens
	Circulation Manager	Mike Schroeder
	Associate Publisher	Mark Kimbell
	VP/Publisher	Mark Crane
	President/CEO	J. Thomas Cottingham

Address: Please send tips, special requests, and other correspondence to

The Editor, *Inside Visual dBASE*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
E-mail address: visual_dbase_win@merlin.cobb.zd.com

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220
E-mail address: customer_relations@merlin.cobb.zd.com

Phone:	Toll free US	(800) 223-8720
	Toll free UK	(0800) 961897
	Local	(502) 493-3300
	Customer Relations Fax	(502) 491-8050
	Editorial Department Fax	(502) 491-3433

Back Issues: To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $8.50 each, $8.95 outside the US. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you.

Copyright: Copyright © 1996, The Cobb Group. All rights reserved. *Inside Visual dBASE* is an independently produced publication of The Cobb Group. The Cobb Group reserves the right, with respect to submissions, to revise, republish, and authorize its readers to use the tips submitted for personal and commercial use.

The Cobb Group and its logo are registered trademarks of Ziff-Davis Publishing Company. *Inside Visual dBASE* is a trademark of Ziff-Davis Publishing Company. dBASE, dBASE IV, dBASE for Windows, and Visual dBASE are registered trademarks of Borland International. Windows is a registered trademark of Microsoft.

Postmaster: Second Class Postage Paid in Louisville, KY.

Postmaster: Send address changes to:

Inside Visual dBASE
P.O. Box 35160
Louisville, KY 40232

Advertising: For information about advertising in Cobb Group journals, contact Tracee Bell Troutt at (800) 223-8720, ext. 430.

Bulk Sales: For information about bulk/group subscription sales, please contact Customer Relations at (800) 223-8720.

Prices:	Domestic	$79/yr ($8.50 each)
	Outside US	$89/yr ($8.95 each)

SECOND CLASS MAIL

dBASE recognizes a special syntax within a character string. Key label names and even numeric constants enclosed within curly braces are interpreted to be the named keystroke or ASCII character. So, the command

```
KEYBOARD("{uparrow}")
```

places an up arrow keypress into the keyboard buffer.

The string you pass to the KEYBOARD( ) function can contain as many of these special constants as you care to include. With this in mind, create a Ditto function for the [Shift][F8] key combination with the following command

```
on key label shift-f8
    keyboard("{uparrow}{ctrl-c}
    {dnarrow}{ctrl-y}{ctrl-v}")
```

This command automates the navigate-copy-navigate-paste process.

After executing this command, you simply press [Shift][F8]. When you do, Visual dBASE plays the keystrokes you defined by the ON KEY LABEL command. First, it presses the [⬆] key to navigate to the previous record. Then the program presses [Ctrl]C, the shortcut key for copying data to the Clipboard.

Next, the routine presses the [⬇] key to navigate back to the current record. The routine presses [Ctrl]Y to erase the contents of the current field. Finally the [Ctrl]V keypress—the shortcut for pasting data from the Clipboard—places the copied value into the current field.

To try this technique, load Visual dBASE, USE any file, and then enter the ON KEY command as listed above. Then, enter the BROWSE command. Navigate to any field in the database and press [Shift][F8]. When you do, Visual dBASE will copy the field value from the previous record into the current record.

Implementing the new Ditto

Our simulated Ditto function works well in BROWSE mode. However, you probably seldom use the BROWSE command in your applications. Even so, you can place a Browse control on a form to provide more control over the look and behavior of the browsing grid.

Suppose you want to activate the new Ditto function when the Browse control is active but deactivate it at other times. To do so, you simply add the appropriate ON KEY statements to the control's OnGotFocus and OnLostFocus event handlers. **Listing A** shows a custom control that demonstrates the technique.

When you customize your environment with this custom control, the procedure will run the procedure Ditto_ON and activate the new Ditto function when the Browse object has focus. That is, the key combination [Shift][F8] will trigger the Ditto_ON procedure. When the Browse object loses focus, the Ditto_OFF procedure deactivates the Ditto function.

Listing A: *A Ditto-enabled Browse control*

```
* DITTOBR.CC
*   Run SET PROCEDURE TO DITTOBR.CC  ADDITIVE to use this
*   custom control in the Form Designer, or select "Setup Custom Controls"
*   from the file menu while using the Form Designer
*
CLASS dittobrowse(FormObj,Name) OF BROWSE(FormObj,Name) Custom
   this.OnGotFocus  = CLASS::Ditto_ON
   this.OnLostFocus = CLASS::Ditto_OFF

   Procedure Ditto_ON
      on key label shift-f8 keyboard("{uparrow}{ctrl-c};
      {dnarrow}{ctrl-y}{ctrl-v}")

   Procedure Ditto_OFF
      on key label shift-f8

ENDCLASS
```

To be continued...

The simple KEYBOARD( ) statement works great for BROWSE mode and Browse controls, but doesn't offer much use for other controls. In a future issue, we'll build a custom control that extends the new Ditto function to the other editing controls. ❖

Printed in USA
This journal is printed on recyclable paper.